B-1 代数仕様言語CafeOBJのための拡張可能な前処理系
An Extensible Pre-processor for the CafeOBJ Algebraic Specification Language

浅羽 義之† 中村 正樹† 天野 憲樹† 二木 厚吉†
Yoshiyuki Asaba Masaki Nakamura Noriki Amano Kokichi Futatsugi

1. はじめに

代数仕様言語 CafeOBJ[1] は代数モデル [2] により形式的にソフトウェアの仕様を記述するための言語であり高い記述能力を持つ．しかし，対象とするソフトウェアの問題によっては仕様の記述が煩雑になる．そこで，CafeOBJ の構文を拡張し煩雑な仕様の記述を簡潔にすることを考える．しかし，CafeOBJ の処理系は自由度の高い仕様の記述を許しており，その文法が複雑である．さらに CafeOBJ の処理系は拡張可能な構造として実現されておらず，その構文を拡張することは容易ではない．

そこで本研究では，CafeOBJ の構文を擬似的に拡張する前処理系を提案する．この前処理系を用いた CafeOBJ の構文拡張はモジュールとして構成された拡張構文を前処理系にプラグインすることで実現される．本研究では，このような構文拡張のモジュールをインクリメンタルに追加できる前処理系の実現を目指す．

2. 問題背景

CafeOBJ は強力なモジュール機構，型付け，パターンマッチ機能などを備えており高い記述能力を持つ．しかし，対象とするソフトウェアの問題によっては仕様の記述が煩雑になる．例として，CafeOBJ による高階関数を用いた map 関数を考える．高階関数とは，引数に関数を取る関数であり，map 関数とはリストの要素に対し，引数で渡された関数を適用するものである．しかし，CafeOBJ は高階関数や高階変数などを言語レベルでサポートしていない．そのため，CafeOBJ ではパラメータ付モジュールを用いて擬似的に高階関数を実現する [3]．以下のコードが CafeOBJ によるパラメータ付モジュールを利用した map 関数の定義である．

```
mod* FUN {  [ Elt ]
  f : Elt -> Elt . }
mod! MAP(X :: FUN) {          -- パラメータ付モジュール
  pr(LIST(X))                 -- モジュールの輸入
  op map : List -> List .     -- 操作の宣言
  var E : Elt .               -- 変数宣言
  var L : List .
  eq map(nil) = nil .         -- 等式の定義
  eq map(E :: L) = f(E) :: map(L) .
}
```

CafeOBJ では仕様をモジュール単位で記述する．上記の MAP モジュールはパラメータ付モジュールとして定義され，FUN モジュールをパラメータの仮引数として使用する．パラメータ付モジュールは汎用的なモジュールであり，そのパラメータに特化したモジュールを作り出す．このような FUN モジュールを導入することで，高階関数を擬似的に実現する．FUN モジュールは引数の型と返り値の型が同じである一引数の関数を持つモジュールである．具体的な FUN モジュールの例として，以下の NAT-FUN モジュールを考える．NAT-FUN モジュールは自然数を 2 倍する double 関数を含む．

```
mod! NAT-FUN { pr(NAT)
  op double : Nat -> Nat .
  eq double(E:Nat) = E * 2 .
}
```

上記のモジュールに加え，CafeOBJ では，仮パラメータと実パラメータの対応関係を与えるビューを作成する必要がある．ここでは，MAP モジュールのパラメータに FUN モジュールを割り当てるためのビューを作成する．具体的には，ソート Elt をソート Nat と対応させ，関数 f を関数 double と対応させる．そして，作成したビューを MAP モジュールのパラメータに割り当て，モジュールをインスタンス化する（make コマンド）．最後に red コマンドで map 関数を実行する．これらの手順を以下に示す．

```
view nat-map from FUN to NAT-FUN {  -- ビューの作成
  sort Elt -> Nat,
  op f(E:Elt) -> double(E:Nat)
}
make NAT-MAP (MAP(X <= nat-map)) -- インスタンス化
red map(1 :: 2 :: 3 :: nil) .    -- map 関数の実行
```

この例からわかるように，CafeOBJ で高階関数を扱うにはパラメータ付モジュールの作成，ビューの作成，モジュールのインスタンス化が必要である．さらに，MAP モジュールに渡すモジュールが異なるとそれに対してビューを定義し直す必要がある．

こうした記述をより簡潔にする方法として CafaOBJ の構文拡張が考えられる．例えば，ビューを作成せずに関数を引数として渡すことができるような構文**hmod, hvar, hop, heq** などを考える．これらの拡張構文を用いて map 関数を定義した例が以下である．

```
hmod! MAP(X :: TRIV) {  pr(LIST(X)) .
  hvar F : Elt -> Elt .
  var E : Elt .
  var L : List .
  hop map : (Elt -> Elt) List -> List .
  heq map(F, nil) = nil .
  heq map(F, E :: L) = F(E) :: map(F, L) .
}
red map (double) (1 :: 2 :: 3 :: nil) .
```

拡張構文により高階関数を直接記述する（double 関数を引数とする）ことが可能になり，記述が簡潔になる．さらに，ビューの作成も不要となる．

3. 本研究のアプローチと前処理系の概要

本研究では CafeOBJ の構文を拡張するために以下のアプローチを採る．

†北陸先端科学技術大学院大学 情報科学研究科, Graduate School of Information Science, Japan Advanced Institute of Science and Technology

- 前処理系による CafeOBJ の擬似的な構文拡張.
- 前処理系の文法は CafeOBJ のサブセットとする.
- CafeOBJ の構文拡張をモジュール単位で実現し（以下，拡張モジュール），拡張モジュールを前処理系にプラグインすることで CafeOBJ の擬似的な構文拡張を実現する.
- CafeOBJ のモジュール機構を利用して拡張モジュールを記述する.

CafeOBJ の処理系は自由度の高い仕様の記述を許しているため，複雑な構文解析処理を行っている．さらに，CafeOBJ の処理系は拡張可能な構造として実現されていない．以上から，CafeOBJ の処理系を直接修正することは容易ではない．したがって，本研究では CafeOBJ に前処理系を導入することで，その構文を擬似的に拡張するアプローチを採る．

次に，前処理系の文法を CafeOBJ のサブセットとする．自由度の高い仕様記述をある程度制限することで，前処理系の構造をシンプルに保ち，その拡張を容易にする．

また，前処理系を用いた CafeOBJ の（擬似的な）構文拡張は構文ごとにモジュール化され，インクリメンタルに追加できることが望まれる．各構文を独立したモジュール（拡張モジュール）として実現することで，その再利用性が向上する．前処理系に拡張モジュールを追加することで，ユーザは前処理系を自由にカスタマイズすることが可能になる．このような前処理系の拡張は CafeOBJ の構文拡張につながる．

さらに，拡張モジュールの記述は CafeOBJ を用いて行う．CafeOBJ で拡張モジュールを記述する利点は構文拡張時に CafeOBJ のモジュール機構の恩恵（パラメータ付モジュールやモジュールの輸入など）を受けることができる点にある．

本研究で提案する前処理系を用いた CafeOBJ の（擬似的な）構文拡張は以下のように実現される．前処理系のユーザは作成した拡張モジュール（CafeOBJ の拡張構文）を前処理系にプラグインする．前処理系は拡張構文を用いて記述された CafeOBJ の仕様を入力とし，それを純粋な CafeOBJ の仕様へ変換して出力する (図 1).

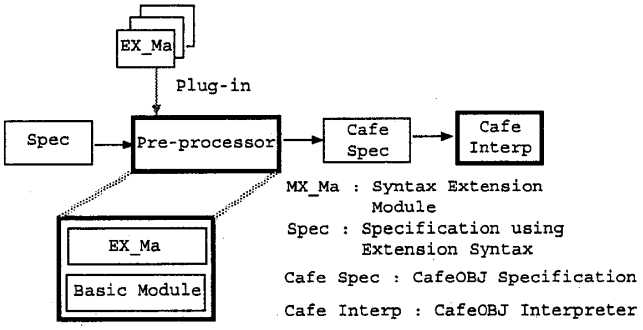

図 1: 前処理系の概要

このような前処理系は拡張モジュールと基本モジュールから構成される（図 1）．基本モジュール（BASIC-SYNTAX）は前処理系の文法（CafeOBJ のサブセット）を定義し，その構文解析を行い，純粋な CafeOBJ のコードを生成する．拡張モジュールは前処理系の構文拡張記述であり，基本モジュールを内包して，前処理系の構文を拡張する．

3.1 拡張モジュールの記述例

拡張モジュールの記述例として，前節の map 関数（高階関数版）に導入した構文**hmod, hvar, hop, heq** を以下にあげる．それぞれの意味は，高階モジュール，高階変数，高階関数，高階のソートを含む等式とする．以下のコードが高階関数を扱う構文（拡張構文）を追加するモジュールの例である．基本モジュールを ex 構文を用いて拡張モジュールに取り込み，op 宣言で拡張構文の文法を定義する．拡張構文を CafeOBJ の構文へ変換するルールは等式で記述する．

```
mod! EX-SYNTAX {  ex(BASIC-SYNTAX) . -- Basic Module
  [ HOrderModule < Module ]   ...
  op (hmod!_{_}) : Symbol ModElement -> HOrderModule .
  op (hvar _:_->_.) : Symbol Sort Sort -> HOrderVar .
  op (hop _:_->_.) : Symbol Arity Sort -> HOrderOp .
  op (heq _=_.) : Term Term -> HOrderEQ .
  eq translate(hmod! S:Symbol { M:ModElement }) = ...
}
```

4. 今後の予定

今後の予定として，まず前処理系の核となる基本モジュールの設計実装が挙げられる．次に，拡張モジュール間の依存関係や衝突について対処する必要がある．これらの問題については，モジュールベースの拡張可能な前処理系 EPP[4] などを参考にする．その他の関連研究として Maude[5] がある．Maude は自己反映計算にもとづく代数仕様言語であり，自己反映計算による構文の拡張を実現している．こうした関連研究の調査を通して得られた知見を前処理系の設計実装にフィードバックする．

参考文献

[1] R. Diaconescu and K. Futatsugi. CafeOBJ Report, AMAST Series in computing 6, World Scientific, 1998.

[2] 二木厚吉, 代数モデルの基礎. コンピュータソフトウェア pp4-22 Vol.13, No1 (1996), ソフトウェア科学会, 1996.

[3] J.A. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. Introducig OBJ. In J. Goguen and G. Malcolm, editors, Software Engineering with OBJ: algebraic specification in action, Kluwer, 2000.

[4] 一杉裕志. 高いモジュラリティと拡張性を持つ構文解析器, 情報処理学会プログラミング研究会論文誌, 1998.

[5] Maude, http://maude.csl.sri.com